University of Glasgow

School of Computing Science

60 YEARS OF COMPUTING AT GLASGOW

Hardware Trends: Implications for Systems Programming

Advanced Operating Systems

Lecture 4

Colin Perkins | https://csperkins.org/ | Copyright © 2017 | This work is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

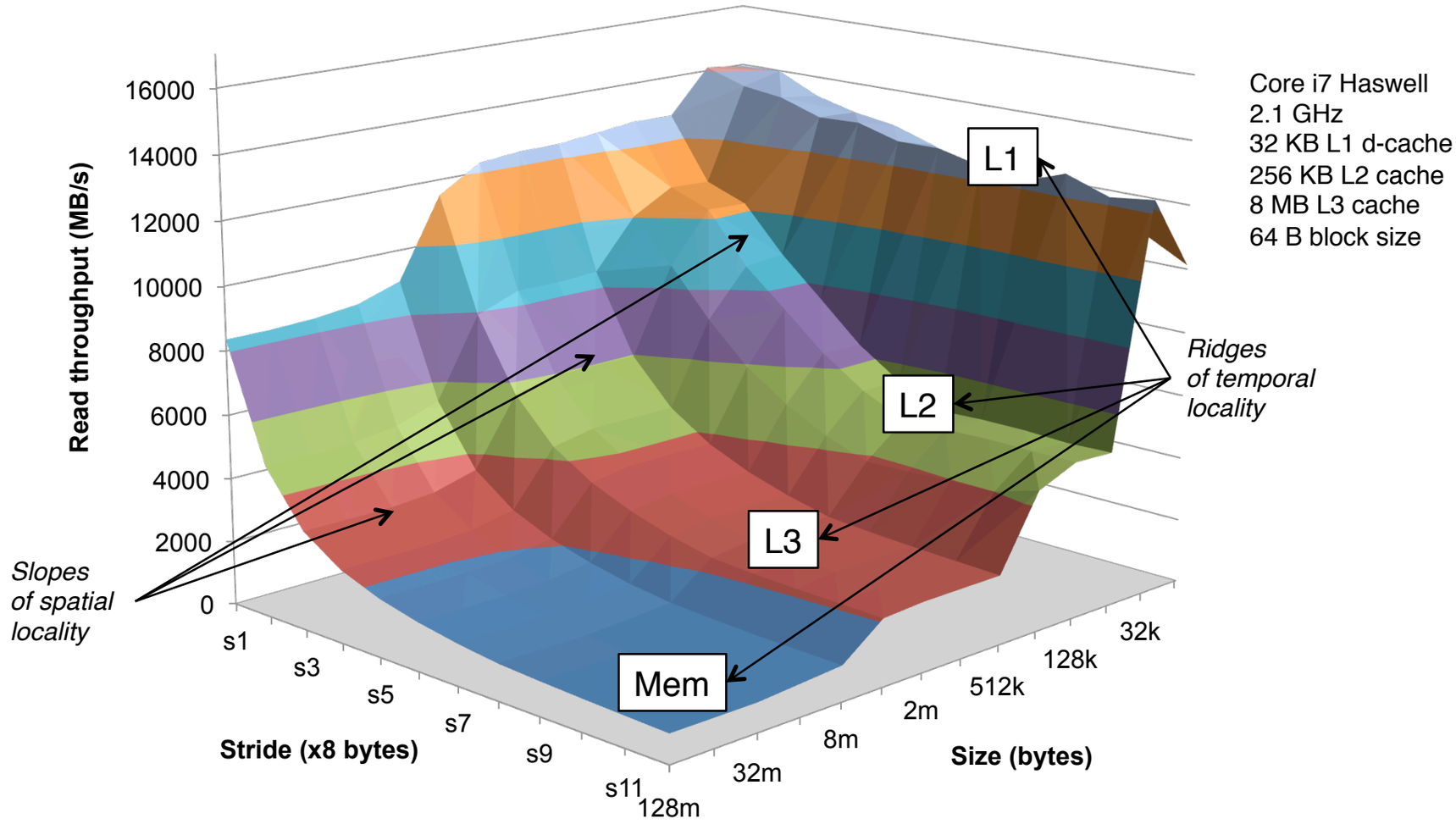
Lecture Outline

- What are implications of hardware changes for systems programming?
  - Memory hierarchy
  - Non-uniform memory access
  - Concurrency
  - Solid state storage
  - High-performance networking

BY ND

Implications of Memory Hierarchy

- Memory isn't flat → access time depends on data location



Smaller values of **stride** indicate data with better spatial locality; **size** is the total amount of data accessed

Source: Bryant & O'Hallaron, "Computer Systems: A Programmer's Perspective", 3rd Edition, Pearson, 2016
Fig. 6.41. http://csapp.cs.cmu.edu/3e/figures.html (website grants permission for lecture use with attribution)

CC BY ND

Implications of Memory Hierarchy

- Data should be packed so objects accessed close in time are located close in memory
  - RAM = Random Access Memory – *but not every access takes equal time*
- Importance of caching and data layout
  - Bringing an object into CPU cache also brings in adjacent objects
    - "cache line size" → 64 bytes, typically
  - Reading from main memory >1000x slower than reading from L1 cache, (generally – highly system dependent)
- Implications of pointer chasing on performance
- Choice of programming language matters
  - A systems language must allow precise control over data layout, memory allocation, and use of pointers – C, C++, Rust, but not Java, C#, etc.

CC BY ND

Implications of NUMA

- Traditional hardware gives appearance of a uniform flat memory, shared between cores
  - Complex cache coherency protocols and memory models
  - Varying degrees of success in hiding the diversity
- Increasingly an illusion, maintained by underlying inter-core network
  - e.g., AMD HyperTransport, Intel QuickPath
  - Point-to-point communications/switching network with message passing protocol

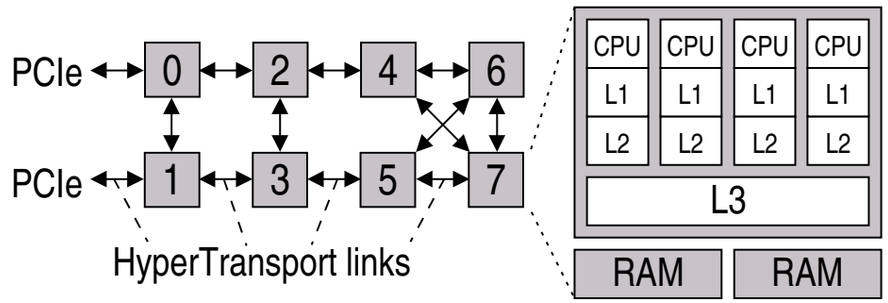


Figure 2: Node layout of an 8×4-core AMD system

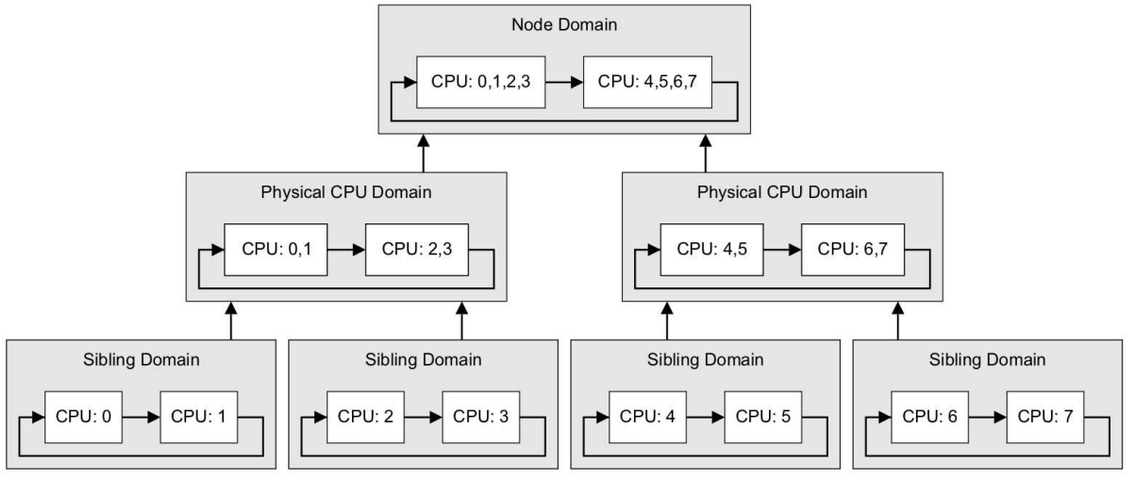
Baumann *et al*, "The Multikernel: A new OS architecture for scalable multicore systems", Proc. ACM SOSP 2009. DOI 10.1145/1629575.1629579

- Processors increasingly exposing heterogeneity – how should it be visible to software?

CC BY ND

Implications of NUMA: Scheduling

- Scheduler must be aware of CPU topology and memory hierarchy to assign threads to physically close processors
  - Avoid blocking on long-distance memory access delays
- Three layers of topology in Linux
  - Hyper-threading within a core (siblings)
  - Cores within a physical CPU package
  - Physical CPUs within a NUMA node
- Load balancing between nodes essential for performance
  - CPU intensive tasks should be separate
  - Communicating threads should be close
  - Monitor and periodically rebalance – heuristic driven, hard to formulate a general policy
  - Linux has `sched_setaffinity()` to bind thread to a particular set of CPUs, to allow manual optimisation



M. Bligh *et al.*, Linux on NUMA systems. Proc. Ottawa Linux Symposium, July 2004

Colin Perkins | https://csperkins.org/ | Copyright © 2017

Implications of NUMA: Memory Allocation

- Locality-aware memory allocation
  - Memory is discontiguous between nodes – partitioned address space
  - Threads should allocate memory local to the node on which they execute; essentially an independent memory management subsystem per node
    - The `malloc()` API is not sufficient in itself – cannot ensure that related data accessed by multiple threads is allocated in memory that is located on the same node, and cannot place allocations on particular nodes (pinning threads to particular nodes can help here)
- Replication of kernel memory on multiple cores
  - Read-only memory regions accessed by multiple threads should be replicated across nodes
    - e.g., the kernel code, shared libraries such as libc
  - Requires support from VM system, to map a virtual address to a different physical address on each core
  - Un-copy-on-write to collapse replicas down to single page if write occurs

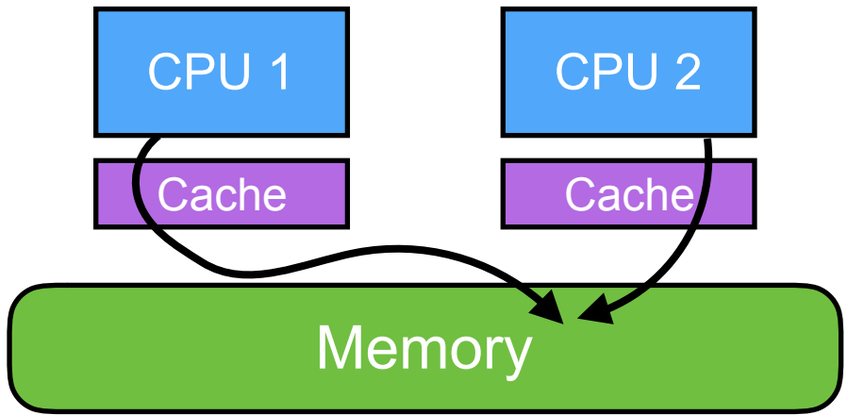
CC BY ND

Implications of Concurrency

- Issues with NUMA closely tied to those with concurrency

- Typical concurrent programming model: threads with shared mutable state
  - Note: introduction of threads into Unix was controversial – this isn't much appreciated now

- Difficult on multicore systems – caches give each core a different view of memory state
  - If reading from cache, potential for other core to have a different value cached
  - If reading from memory, latency penalty counters benefit of parallelism



CC BY ND

Cache Coherence in Concurrent Systems

- Shared mutable state → processors must have a coherent view of memory
  - At least, at synchronisation points defined by the language memory model
  - Potentially significant performance gains to be achieved by partitioning memory between processors
  - Scheduling threads that share memory on processors that share cache can result in significant speedup; equally – significant slowdowns occur when unrelated data sharing a cache line is accessed by multiple cores
    - Access by one core invalidates the cache, causing a flush to main memory and reload; the other core then accesses, and the data is flushed back – ping-pong occurs
    - Can causes slowdowns of many orders of magnitude
- Significant implementation complexity
- Complex programming model

CC BY ND

Is a Shared Memory Model Reasonable?

- Which is cheaper – message passing or shared memory?
  - Graph shows shared memory costs (1-8 cores, SHM1...SHM8) and message passing costs (MSG1...MSG8) for a 4 x quad-core server, with AMD HyperTransport
  - Cost of cache coherency protocols increases with the number of cores – messages can be cheaper, depending on the architecture
- Which is easier to program?
  - Shared-state concurrency is notoriously hard to program (locks, etc.)
  - Systems that avoid shared mutable state are frequently cited as easier to reason about
- For how long will it be reasonable to maintain illusion of shared memory?
  - For the hardware?
  - For the programmer?

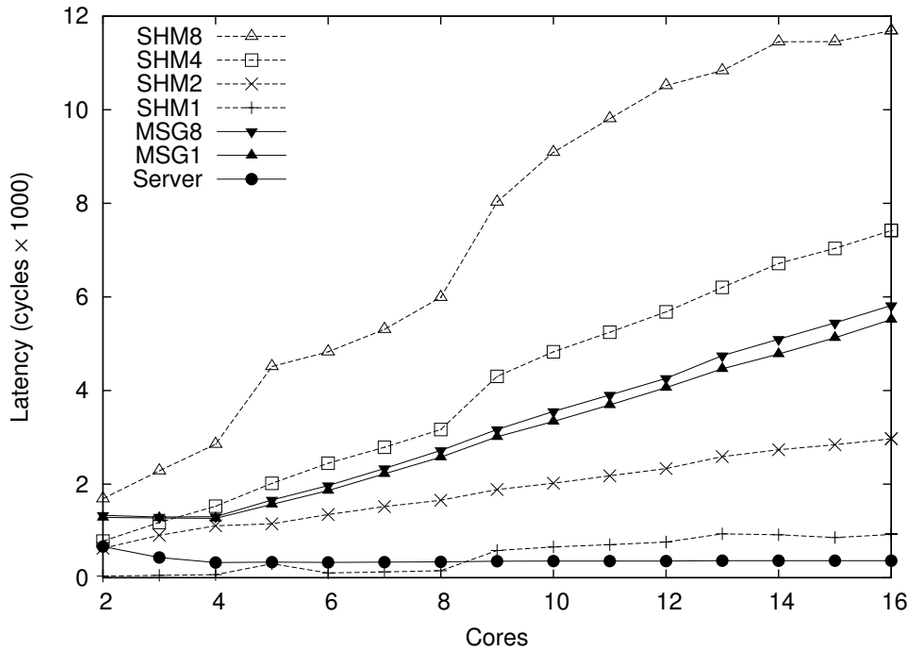


Figure 3: Comparison of the cost of updating shared state using shared memory and message passing.

Baumann *et al.*, "The Multikernel: A new OS architecture for scalable multicore systems", Proc. ACM SOSP 2009. DOI 10.1145/1629575.1629579

See also: F. Gaud *et* al., "Challenges of Memory Management on Modern NUMA Systems", Communications of the ACM, Vol. 58, Num.12, pp.59-66. Dec. 2015, DOI: 10.1145/2814328

CC BY ND

Implications of Solid State Storage

- Properties of storage are changing
  - Hard disks → solid state disks
  - Importance of data layout changing:
    - Was a performance optimisation
    - Becoming critical for correctness

CC BY ND

Behaviour of hard disks

- Electro-mechanical device
  - Spinning disk platters
  - Moving read/write heads
- Seek time, rotation → latency
- Poor random access performance: layout of data on disk important for good performance
  - File systems seek to avoid fragmentation within a file
  - Place frequently accessed files physically close to each other

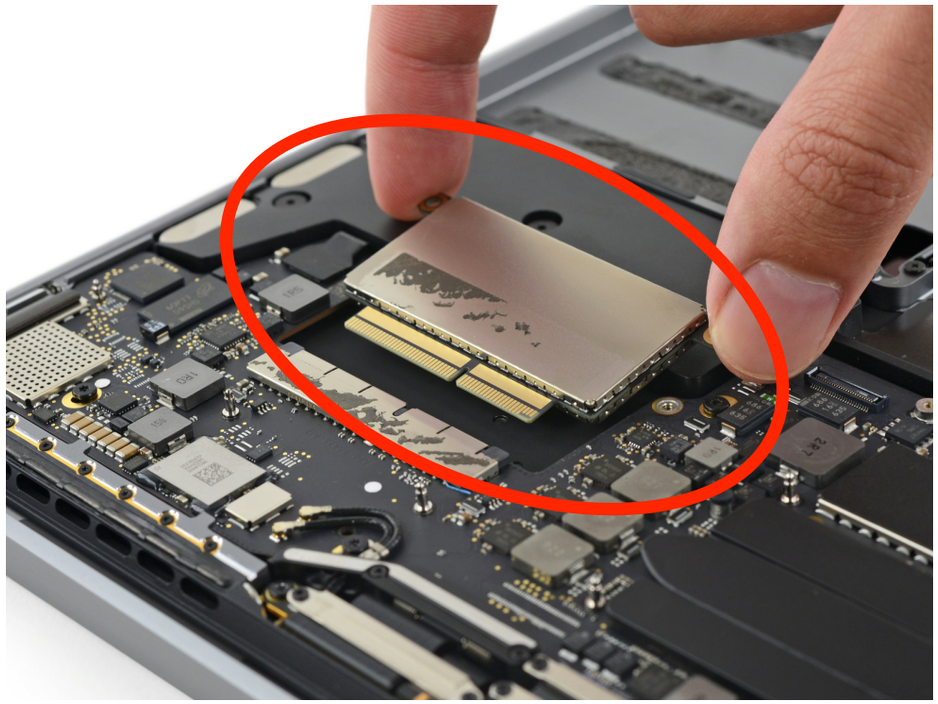


- Similar issues to the memory cache hierarchy, with much higher latency

CC BY ND

Behaviour of solid state disks

- Non-volatile flash memory – no moving parts
- Much lower latency – much better random access performance
  - No need to wait for physical rotation of disk or movement of the heads
  - Location of files on disk unimportant for performance – file system complexity can be reduced
- Limited write cycles per block
  - Physical location of data on SSD matters for correctness – repeatedly writing to the same block will cause failures
  - Usually handled by SSD firmware without knowledge of the operating system
    - Firmware complexity becoming an issue – SSDs essentially run their own operating systems
    - Beneficial for operating system to tell the SSD “this data is deleted” rather than just discard it



Source: iFixit/CC-BY-NC-SA 3.0

CC BY ND

Implications for filesystems

- Most existing filesystems spend considerable effort to ensure good performance
  - Optimising placement of files
  - Avoiding fragmentation
- Now largely unnecessary with solid state storage
- Data integrity with limited write cycles now a more critical issue for filesystem design

CC BY ND

Implications of Performant Networking

- Modern network interfaces outperform the network stack
  - Multiple receive queues for concurrent connections
  - DMA to several cores simultaneously
  - Can process small packets at line rate – 64 byte packets on 10Gbps links gives very little time for the operating system to process each packet

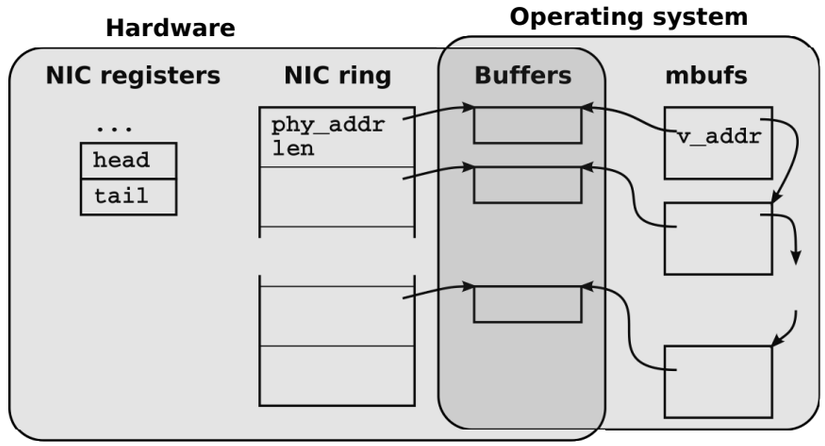


Figure 1: Typical NIC's data structures and their relation with the OS data structures.

Source: L. Rizzo. netmap: a novel framework for fast packet I/O. In Proceedings of the USENIX Annual Technical Conference, Boston, MA, USA, June 2012. USENIX.

- Traditional Berkeley sockets interface, with a system call per packet, cannot deliver data at required rate
- Alternatives:
  - Operating system bypass – netmap, DPDK
  - New APIs – StackMap

CC BY ND

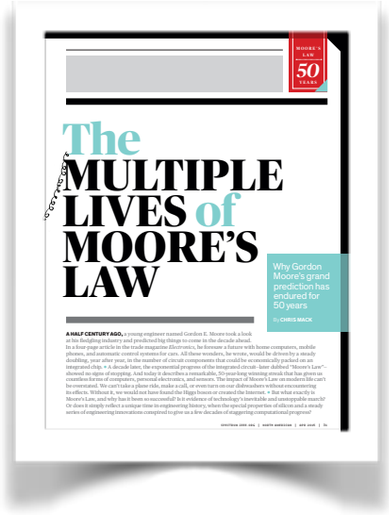
Conclusions

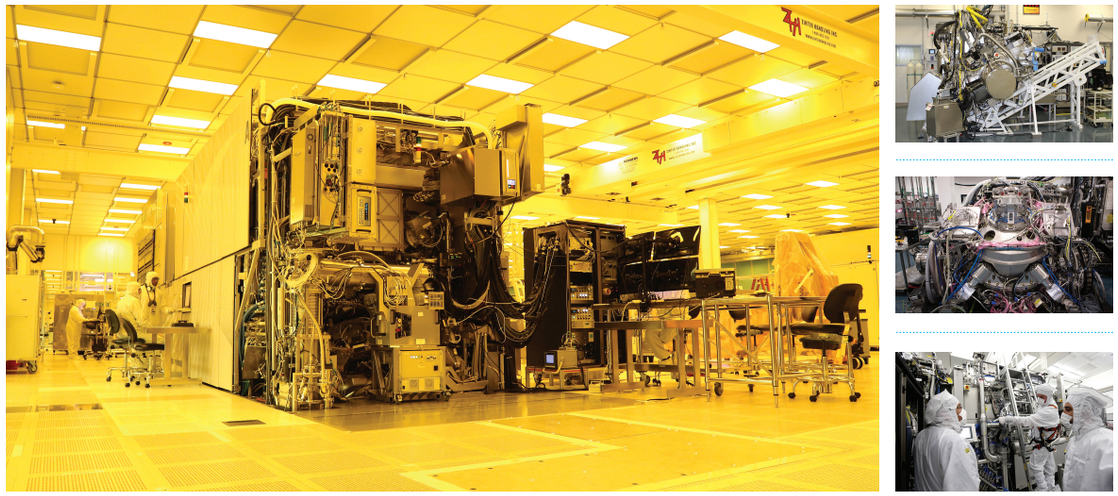
- The way we build operating systems has to change
  - Memory hierarchy, caching, NUMA, concurrency → threading unsuitable
  - Solid state storage → rethink file system internals
  - Performant networking → rethink network APIs and protocol design

Colin Perkins | https://csperkins.org/ | Copyright © 2017

Further Reading

- C. Mack, “The Multiple Lives of Moore’s Law”, IEEE Spectrum, Vol. 52, Num. 4, April 2015. DOI: 10.1109/MSPEC.2015.7065415
- R. Courtland. The molten tin solution. IEEE Spectrum, 53(11):28–41, November 2016. DOI: 10.1109/MSPEC.2016.7607024





The Molten Tin Solution

BY RACHEL COURTLAND

After a 30-year journey, EUV chip-printing technology is now poised to help keep Moore’s Law on track

PUTTING EUV TO THE TEST: This EUV scanner (an ASML NXE:3300B) is used to print chip features at a SUNY Polytechnic Institute facility in Albany, N.Y. The EUV light needed to expose wafers is created near the bottom of the scanner, on the side visible in the foreground of this photograph. The far end of the machine is attached to a “track” that coats the wafers before exposure and processes them once they are done.

INSIDE THE MACHINE: To generate EUV, pulses of CO_2 laser light are sent into a vessel [top and middle] where they collide with tiny tin droplets to create plasma. This partially assembled EUV scanner [bottom] at ASML’s headquarters in Veldhoven, Netherlands, is one of the company’s more recent models.

28 | NOV 2016 | NORTH AMERICAN | SPECTRUM.IEEE.ORG

LEFT: IBM RESEARCH; RIGHT: ASML (3)

SPECTRUM.IEEE.ORG | NORTH AMERICAN | NOV 2016 | 29

Colin Perkins | https://csperkins.org/ | Copyright © 2017